MISB MOTION IMAGERY STANDARDS BOARD	MISB ST 1204.2
STANDARD	
Motion Imagery Identification System (MIIS) – Core Identifier	**27 June 2019**

1 Scope

Many different sensors produce Motion Imagery Data distributed across many different networks and received by many different users and systems. Assigning a consistent name, label, or identity to each Motion Imagery source helps to coordinate analysis, manage assets, and avoid confusion.

The Motion Imagery Identification System (MIIS)-Core Identifier addresses this issue. Specifically, this standard defines: (1) required identifiers for a Motion Imagery stream or file; (2) points to insert identifiers as the Motion Imagery Data flows from source to user; and (3) methods and formats for inserting identifiers into a Motion Imagery stream or file. This standard also provides guidance on extracting Universal Unique Identifiers (UUID) for use as enterprise identifiers of Motion Imagery systems.

MIIS-Augmentation Identifiers provide a framework for including human-readable supplemental information to Motion Imagery Data to better manage and exploit Motion Imagery. MISB ST 1301 [1] defines and documents Augmentation Identifiers.

The MIIS solves four problems:

(1) Determining whether two (or more) Motion Imagery streams or files are from the same source (sensor/platform).
(2) Determining whether two (or more) Motion Imagery streams or files are from two different sources.
(3) Basis for pedigree information about the Motion Imagery (i.e., tracking all Motion Imagery manipulations from source through receiver).
(4) Linking useful identifying information about the Motion Imagery stream or file to the Motion Imagery source.

When fully implemented, the MIIS fulfills the need to provide a consistent and unique identifier for all sensors and platforms.

2 References

[1] MISB ST 1301.2 Motion Imagery Identification System - Augmentation Identifiers Local Set, Feb 2014.

[2] ISO/IEC 14977:1996 Information technology -- Syntactic metalanguage -- Extended BNF.

[3] SMPTE ST 336:2017 Data Encoding Protocol Using Key-Length-Value.

[4] "XML Technology Schema," 2015. [Online]. Available: https://www.w3.org/standards/xml/schema.

[5] ISO/IEC 8825-1:2015 Information Technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER).

[6] MISB ST 0807.23 MISB KLV Metadata Registry, Feb 2019.

[7] MISB MISP-2019.3 Motion Imagery Standards Profile, Jun 2019.

[8] MIL-STD-1472G Department of Defense - Design Critera Standard: Human Engineering, 11 Jan 2012.

[9] ITU-T X.667 | ISO/IEC 9834-8 Procedures for the generation of universally unique identifiers (UUIDs) and for their use in the international object identifier tree under the joint UUID arc, 14 Oct 2012.

[10] IETF RFC 4122 A Universally Unique IDentifier (UUID) URN Namespace, Jul 2005.

[11] NGA.RP.0001_1.0.0 NGA Recommended Practice for Universally Unique Identifiers, Jan 2013.

[12] NIST SP 800-90A Rev. 1 Recommendation for Random Number Generation Using Deterministic Random Bit Generators, Jun 2015.

[13] *A check digit system for hexadecimal numbers, Dr. Markku et al.*.Department of Mathematical Sciences, University of Oulu, 90014 Oulu, Finland.

[14] *On some properties of a check digit system, Dr. Markku Niemenmaa.*2012 IEEE International Symposium on Information Theory Proceedings.

3 Acronyms

EBNF Extended Backus–Naur Form
FCID Foundational Core Identifier
GCS Ground Control Station
ID Identifier
IEEE Institute of Electrical and Electronics Engineers
KLV Key Length Value
LVMI Large Volume Motion Imagery
MCID Minor Core Identifier
MIIS Motion Imagery Identification System
MISB Motion Imagery Standards Board
MISP Motion Imagery Standards Profile
NIST National Institute of Standards and Technology
NGA National Geospatial-Intelligence Agency
NSG National System for Geospatial-Intelligence
RP Recommended Practice
SMPTE Society of Motion Picture & Television Engineers

ST	Standard
UAS	Unmanned Airborne System
UUID	Universally Unique Identifier
XML	eXtensible Markup Language

4 Revision History

Revision	Date	Summary of Changes
ST 1204.2	06/27/2019	• Core Identifier "Version" changed to align with change in Core Identifier template itself, not to a version of this standard • Req's -02, -12, -16, -20, -25, -26, -27, -30, -31, -32, -43 reworded for clarity; Req -25: removed "reported and"; Added Req -45 • Deprecated requirements -33, -35, -36, -37, -38 as captured in the MISP and other supporting MISB standards • Changed Table 1 & 2 captions and headings for consistency • Removed VANC from Table 4 • Removed section references in requirements • Text changes for clarity; changed "compliant" to "conformant" • Added examples for Usage Value in Section 7.3.1 • Updated algorithm in Appendix B • Added additional examples of Core Identifier in new Appendix E • Updated references; some deleted as no longer needed

5 Overview

Many platforms (sources) produce Motion Imagery Data and many systems (users) receive this data. Figure 1 illustrates the ecosystem of producers, dissemination, and the receivers for Motion Imagery assets.

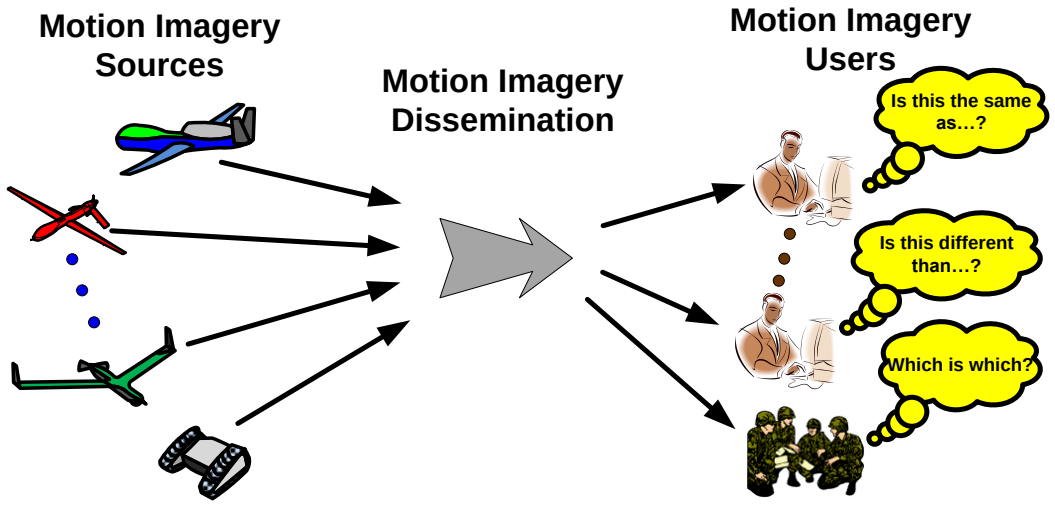

Figure 1: Conceptual Data Flow

The users of Motion Imagery include: Sensor operators, Soldiers, Mission Coordinators, Exploiters, Production personnel, Photogrammetrists, Historians, Archivists, Security, and Lawyers. A Motion Imagery Identifier facilitates the coordination of activities across the entire (world-wide) user base. Providing the identity of the Motion Imagery source data using a **Core Identifier** plus any additional contextual information with **Augmentation Identifiers** achieves this goal. A Core Identifier is a unique "name" for a Motion Imagery source. An Augmentation Identifier provides contextual information, which is information about the data source and its usage (e.g., Mission IDs, ATO's, feed color). The MIIS is a composite of both the Core Identifier and Augmentation Identifiers as shown in Figure 2.

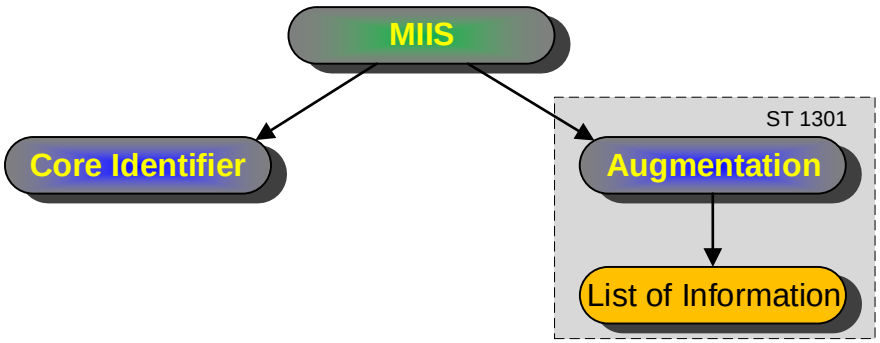

Figure 2: MIIS Identifiers

Augmentation Identifiers apply to specific situations and applications, so there is no required list of items specified. They do however provide valuable additional information to qualify Motion Imagery Data. MISB ST 1301 defines and documents the use of Augmentation Identifiers.

Although Motion Imagery Data exists in both analog and digital form, analog Motion Imagery does not support the necessary metadata carriage for this standard. Digital Motion Imagery affords the inclusion of identifiers directly to imagery frames and accompanying metadata. Thus, this standard applies to digital Motion Imagery only.

Requirement	
ST 1204.1-01	All MIIS conformant Digital Motion Imagery Data shall contain a Core Identifier in either the imagery frames or metadata or both.

6 Core Identifier

A Core Identifier is a collection of up to three **Identifier Components** combined to form a unique name for the Motion Imagery Data. An Identifier Component is a UUID generated either during the manufacture of a device, or on an as-needed basis throughout the data flow and inserted at different points during the creation and/or dissemination of the Motion Imagery Data.

"Generated" means to create a UUID either from unique device information such as serial numbers, model numbers, etc., or from a random number generator (described in Appendix A). Inserting Identifier Components means to include the Identifier Components into the Motion Imagery Data consistent with the format of the Motion Imagery within its transport container. Identifier Components are either created or added to an existing Core Identifier. Ideally, at

insertion the identifier will be frame accurate which is important for platforms with multiple sensors where switching among sensors may occur at frame boundaries.

There are two types of Core Identifiers: **Foundational** and **Minor.** A Foundational Core Identifier is composed of up to three Identifier Components: Sensor Device Identifier, Platform Device Identifier, and Window Identifier. Constructing a Foundational Core Identifier requires a device and/or Window Identifiers be known, which limits their insertion to either on-board the platform or at a ground control station (e.g., GCS for a UAS). A Minor Core Identifier is a single Identifier Component from a randomly generated UUID and created/inserted *after* the platform; not on-board a platform. Figure 3 illustrates the Core Identifier and its types.

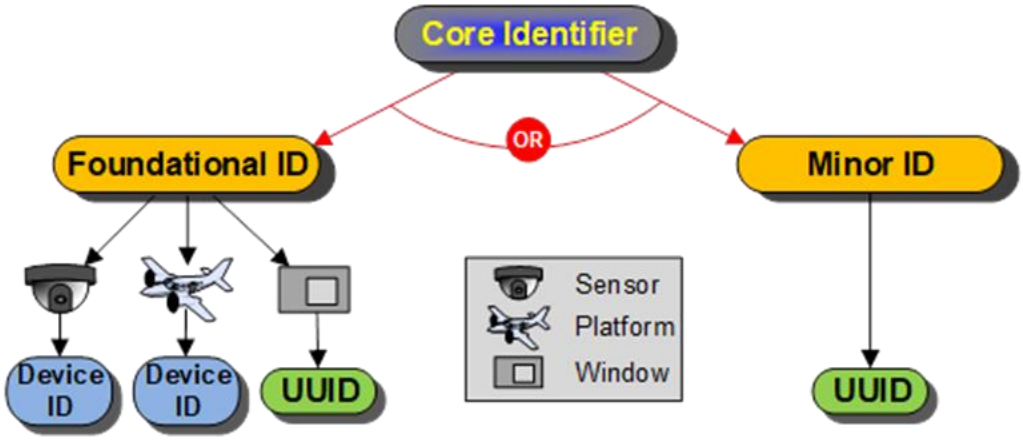

Figure 3: Core Identifier Types

The quality of a Core Identifier depends on where in the data flow the Identifier Components get generated and inserted. A high-quality Core Identifier means it is the same identifier across all users of the Motion Imagery (i.e., ubiquitous); it is unique across all sensors/platforms; and the physical data source (i.e., sensor, platform and optional window) can be determined from the identifier. A low-quality Core Identifier only provides the same identifier for a subset of users, typically at a single site. Inserting an identifier as early as possible into the Motion Imagery Data produces a higher identifier quality level. For example, generating and inserting Identifier Components on-board the Sensor/Platform ensures a high-quality Core Identifier because every user will have the same identifier. Alternatively, generating and inserting a Core Identifier at a user's site (at the user end of the data flow) produces a low-quality Core Identifier because each end user group will have a different identifier for the same Motion Imagery.

The Identifier Components include information on the insertion point in a data stream, so users of the identifiers understand the Core Identifier quality. There are four basic generation/insertion points: (1) automatically within the sensor/platform; (2) on-board the platform from a host computer (e.g., generated by flight computer, inserted by encoder); (3) within a control station; and (4) any point from the control station to the end user. Identifier Components include one of these four identifier quality values, so that end users have knowledge on the origin of the identifier.

A Core Identifier should have the following important properties:

1) **Unique** - no two Core Identifiers are the same amongst any data sources

2) **Compact** – bit-efficient because they repeat in the stream frequently
3) **Anonymous** - the source of Identifier Components is untraceable
4) **Automatic** - Identifier Components are independent from human generation

Foundational Core Identifiers should also have the following properties:

5) **Persistent** - the same Identifier Components apply after a system is power cycled
6) **Consistent** - the same Identifier Components apply to a specific device even if it after separated from one host device and connected to a different host device
7) **Ubiquitous** - the Core Identifier is the same for all users to determine similarity or distinctness
8) **Frame Accurate** - the Core Identifier is discoverable in every frame of the imagery
9) **Sensor Identification** - the Core Identifier enables the determination of which sensor is in use
10) **Platform Identification** - the Core Identifier enables the determination of which platform is in use

In order to support the types of users identified above, adhering to all ten of these properties is necessary when generating a Core Identifier.

6.1 Foundational Core Identifier

A Foundational Core Identifier provides a well-defined, unique, persistent identifier for a Motion Imagery Data source. The insertion of a Foundational Core Identifier is either on-board a platform, before any storage or transmission of the Motion Imagery from a platform, or within the control station of the sensor/platform.

A Foundation Core Identifier includes three Identifier Components: Sensor Identifier, Platform Identifier and Window Identifier (shown as icons in Figure 3). The Sensor Identifier and Platform Identifier are Device Identifiers which can be generated and inserted at three different insertion points in the data flow: (1) automatically within the sensor/platform device (**Physical Identifier**), (2) on-board the platform from a host computer (e.g., generated/inserted by flight computer or encoder) (**Virtual Identifier**), or (3) from a control station (**Managed Identifier**). Figure 4 illustrates the Device Identifier and its types.

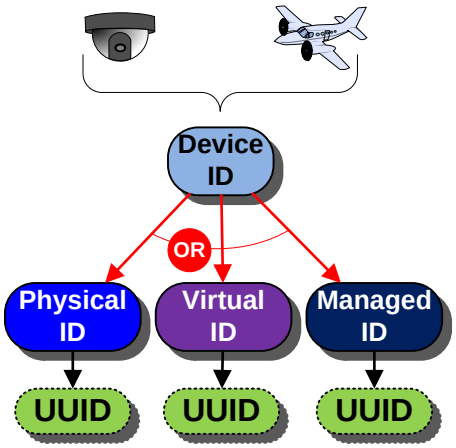

Figure 4: Device Identifier and Types

The Window Identifier in Figure 3 is a UUID generated and inserted for a sub-section of the full Motion Imagery frame. The Window Identifier is not a Device Identifier type.

6.1.1 Device Identifier and Types

As discussed above Sensor Identifiers and Platform Identifiers are Device Identifiers which can be Physical, Virtual, or Managed Identifiers. The following provides more details on these types.

Physical Identifier: generated or stored *within* a device itself and never changes over the lifetime of the device. Should the device be power cycled the exact same identifier applies (i.e., persistent). Likewise, removing the device from the system and later re-integrating (or moved to another system) the same identifier applies for that device (i.e., consistent). Physical Identifiers are frame accurate, meaning every frame of the Motion Imagery Data associates with a Physical Identifier. No device will generate, store or use a predefined Physical Device Identifier assigned to another device.

Virtual Identifier: generated or stored *outside* of the physical device by an external host device (e.g., flight computer) and managed by that host so it is persistent. If the device or host is power cycled the exact same identifier applies (i.e., persistent). Similarly, the host will manage the change to the Virtual Identifier when swapping or changing the device. Inserting a Virtual Identifier into all copies of the Motion Imagery Data sent from, or stored on, the platform maintains the ubiquitous nature of the identifier. Although a Virtual Identifier does not need to be frame accurate, requirement MISB ST 1204.1-12 places limits on its update rate.

Managed Identifier: generated or stored by the control station (e.g., GCS) and managed by the control station so it is persistent. Devices in a control station need to provide the exact same identifier (i.e., persistent) if power cycled. The control station will manage changes to the Managed Identifier when swapping or changing the device. Although a Virtual Identifier does not need to be frame accurate, requirement MISB ST 1204.1-16 places limits on its update rate.

The difference between a Virtual Identifier and Managed Identifier is the Virtual Identifier is ubiquitous while a Managed Identifier only serves users downstream from the control station; this means a Virtual Identifier will be a higher identifier quality level than a Managed Identifier.

Because they automatically insert into the Motion Imagery Data and are less prone to error (i.e., they are plug-n-play), a Physical Identifier is ideal; however, a Physical Identifier may require an upgrade to the device's hardware. Implementing a Virtual or Managed Identifier is likely a software upgrade; however, systems that use them will have greater data management overhead and may rely on human interaction and input, which can be error prone.

Systems can construct a Foundational Core Identifier using different types of Sensor Identifiers and Platform Identifiers. For example, a sensor may generate a Physical Identifier and the platform may generate a Virtual Identifier.

An **Identifier Quality Level** rating (shown in Table 1) for various permutations of Physical, Virtual and Managed Sensor Identifier and Platform Identifier devices provides a gauge on the degree a system meets the goal of this standard. The scale lists these permutations from highest quality to lowest quality with Identifier Quality Level 1 the highest. A goal of this standard is for all systems to implement Identifier Quality Level 1 for the Sensor Identifier and Platform Identifier.

Table 1: Identifier Quality Level vs. Device Identifier Type

Identifier Quality Level	Sensor Identifier Quality Value	Platform Identifier Quality Value
1	Physical	Physical
2	Physical	Virtual
3	Physical	Managed
4	Virtual	Physical
5	Virtual	Virtual
6	Virtual	Managed
7	Managed	Physical
8	Managed	Virtual
9	Managed	Managed

6.1.2 Sensor Identifier

The definition of "Sensor" varies depending on the context used. For this standard the definition of a Sensor is a device that converts some optical (or other) phenomenon into an electrical signal. Based on this, a Sensor Identifier is associated to a focal plane array and its electronics. The Sensor Identifier is not the identifier for the gimbal ball or lens system, because a gimbal may have multiple cameras mounted within them, and, the lens system is not converting optical phenomenon into electrical signals. Identifiers for these other devices, such as the gimbal ball or lens system, are Augmentation Identifiers (see MISB ST 1301).

As discussed in Section 6.1, the Sensor Identifier can be a Physical Identifier, Virtual Identifier, or Managed Device Identifier; denoted by the Sensor-ID-Type.

A Physical Sensor Identifier is an identifier automatically generated (i.e., with no human interaction) and inserted into the Motion Imagery Data by a sensor device.

A Virtual Sensor Identifier is an identifier generated or stored on-board the platform by a host (e.g., flight computer) and inserted into all copies of the Motion Imagery Data before storage and/or transmission. A user can manually enter a Virtual Sensor Identifier, if needed.

A Managed Sensor Identifier is an identifier generated or stored within the control station for the platform/sensor and inserted into the Motion Imagery Data before exploitation within the control station and/or dissemination from the control station.

A Physical Sensor Identifier provides the highest quality identifier, followed by a Virtual Sensor Identifier, and finally the Managed Sensor Identifier which is the lowest identifier quality.

For multi-sensor systems which mosaic multiple Motion Imagery frames into one large frame (e.g., LVMI) or fuse multiple sensor images into one image, refer to Section 6.1.2.1 for requirements and guidance on Multi-Sensor Systems.

A goal of this standard is for systems to provide consistent and persistent identifiers as much as possible. The following requirements promote this goal by ensuring the identifiers only change when necessary.

Requirement(s)	
ST 1204.1-02	The Sensor Identifier shall identify the physical device detecting some optical (or other) phenomenon.
ST 1204.1-03	The Sensor Identifier shall be a UUID generated following the guidelines in Appendix A.
ST 1204.1-04	When MIIS conformant Sensors that provide a digital output are used, the output shall include a Sensor Identifier.
ST 1204.1-05	When a Sensor Identifier is a Physical Sensor Identifier the Sensor-ID-Type shall be “Physical”.
ST 1204.1-06	When a Sensor has a Physical Sensor Identifier, the Sensor shall insert the Physical Sensor Identifier into every frame of uncompressed Motion Imagery Data.
ST 1204.1-07	When a Sensor has a Physical Sensor Identifier and the Sensor is power cycled, the Sensor shall use the same Physical Sensor Identifier.
ST 1204.1-08	When a Sensor has a Physical Sensor Identifier and the Sensor is disconnected from a system and reattached to the same system or alternate system, the Sensor shall use the same Physical Sensor Identifier.
ST 1204.1-09	When a Sensor Identifier is a Virtual Sensor Identifier the Sensor-ID-Type shall be “Virtual”.
ST 1204.1-10	When a system uses a Virtual Sensor Identifier and the Sensor or host is power cycled, the system shall use the same Virtual Sensor Identifier.
ST 1204.1-11	When a system uses a Virtual Sensor Identifier and the Sensor is replaced with a new Sensor, the host shall change or generate a new Virtual Sensor Identifier for the new Sensor.
ST 1204.1-12	Where a system uses a Virtual Sensor Identifier, the host device shall ensure the proper Virtual Sensor Identifier is inserted into the Motion Imagery Data at the correct time (e.g., when a sensor change occurs) to within a system defined value number or fraction of seconds (nominally ½ second) of the change.
ST 1204.1-13	When a Sensor Identifier is a Managed Sensor Identifier the Sensor-ID-Type shall be “Managed”.
ST 1204.1-14	When a system uses a Managed Sensor Identifier and the Sensor or Control station is power cycled, the system shall use the same Managed Sensor Identifier.
ST 1204.1-15	When a system uses a Managed Sensor Identifier and the Sensor is replaced with a new Sensor, the control station shall change or generate a new Managed Sensor Identifier for the new Sensor.
ST 1204.1-16	Where a system uses a Managed Sensor Identifier, the control station shall ensure the proper Managed Sensor Identifier is inserted into the Motion Imagery

	Data at the correct time (e.g., when a sensor change occurs) to within a system defined value number or fraction of seconds (nominally ½ second) of the change.
ST 1204.1-17	When Motion Imagery Data contains a Sensor Identifier no MIIS system shall replace the Sensor Identifier with another Sensor Identifier.

6.1.2.1 Multi-Sensor Systems

Some systems generate frames of Motion Imagery from a collection of sensors, such as Class 2 Motion Imagery LVMI systems or sensor fusing systems. For the resulting composite Motion Imagery to have one consistent Sensor Identifier assigned, the Sensor Identifiers from each Sensor are combined into one Sensor Identifier by using the UUID Hashing method outline in Appendix A.

If the Sensor configuration changes (i.e., a Sensor is swapped out) then the resulting Sensor Identifier hashed value will change as well, indicating to end users or systems that a change was made which could flag the need for updating image calibration, etc. Figure 5 illustrates an example of a multi-sensor data flow. Each Sensor on the left (blue) has its own Sensor Identifier (Sensor ID1 through Sensor ID4). A UUID Hash algorithm combines the IDs to produce Sensor ID-A used both in the mosaic frame data and each individual Motion Imagery stream. Each individual stream extracted (i.e., window) from the full frame data receives a Window Identifier.

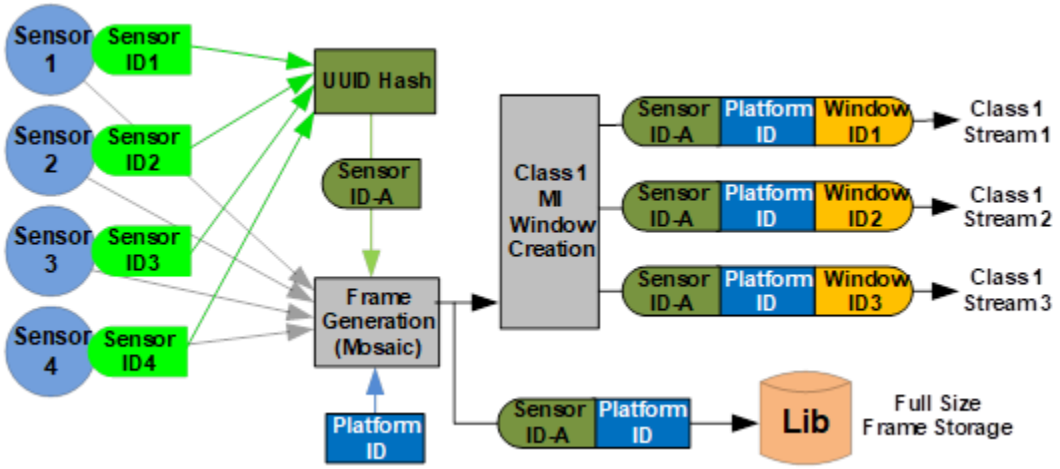

Figure 5: Example Multi-Sensor Data Flow

The UUID Hash will produce a different Sensor ID-A value if there is a change in the sensing system, such as replacing a sensor.

Requirement(s)	
ST 1204.1-18	When a MIIS conformant multiple sensor system is used to form a single Motion Imagery source, the Sensor Identifier's UUIDs from each source shall be combined into one UUID using the UUID Hashing technique from Appendix A.
ST 1204.1-19	When multiple sensors are used to form a single Sensor Identifier, the Sensor-ID-Type shall be the lowest Identifier quality Sensor-ID-Type of the group.

6.1.3 Platform Identifier

Platforms are typically collections of devices grouped together around a physical structure or skeletal frame. Devices such as flight computers, sensors, robotic arms, etc. can change or potentially move to other platforms. To provide consistency, in this standard the definition of a Platform is the physical structure or skeletal frame to which the devices physically connect. This definition prevents the Platform Identifier from moving with one of the devices (i.e., a flight computer moved from one platform to another does not maintain the same identifier).

As discussed in Section 6.1, the **Platform Identifier** can either be a Physical Identifier, Virtual Identifier, or Managed Identifier; denoted by the Platform-ID-Type.

A Physical Platform Identifier is an identifier that is discoverable on-board without human interaction and inserted into all copies of the Motion Imagery Data before storage and/or transmission. As the platform structure or skeletal frame is not an electrical device it will need some form of physical tagging for automatically providing or enabling the generation of the Physical Platform Identifier. For example, with an aircraft the identifier could be based on the serial number of the aircraft frame and automatically scanned during power-up.

A Virtual Platform Identifier is an identifier generated or stored on-board the platform by a host (e.g., flight computer) and inserted into all copies of the Motion Imagery Data before storage and/or transmission. Generating a Virtual Platform Identifier may be through automatic or manual means.

A Managed Platform Identifier is an identifier inserted within the control station of the platform/sensor.

A Physical Platform Identifier provides the highest identifier quality followed by a Virtual Platform Identifier with the Managed Platform Identifier the lowest identifier quality.

A Platform Identifier is unnecessary when a platform is not associated with a sensor. For example, while arguable a handheld camera does not have a platform, so a Platform Identifier is unnecessary. For ground-based static-mounted sensors (i.e., sensors mounted on poles, fences or buildings) the Platform Identifier is optional but recommended.

A goal of this standard is for systems to provide consistent and persistent identifiers as much as possible.

Requirement(s)	
ST 1204.1-20	The Platform Identifier shall identify the physical device (or skeletal frame) to which the sensor is physically attached (i.e., aircraft, UAV, Humvee, etc.).

ST 1204.1-21	The Platform Identifier shall be a UUID generated following the guidelines in Appendix A.
ST 1204.1-22	When a Platform Identifier is a Physical Platform Identifier the Platform-ID-Type shall be “Physical”.
ST 1204.1-23	When a Platform Identifier is a Virtual Platform, the Platform-ID-Type shall be “Virtual”.
ST 1204.1-24	When a Platform Identifier is a Managed Platform Identifier, the Platform-ID-Type shall be “Managed”.
ST 1204.1-25	Platform Identifiers shall be inserted into Motion Imagery Data onboard the Platform.

6.1.4 Window Identifier

The **Window Identifier** identifies an area-of-interest extracted from a Motion Imagery source. This identifier applies only when creating and transmitting sub-images of a Motion Imagery. For example, a LVMI data source may have multiple users extracting windows from the full frame imagery and forming one or more Motion Imagery streams. Each stream would have an identical Sensor Identifier and Platform Identifier, so a Window Identifier enforces the uniqueness requirement. The Window Identifier is a UUID generated following the guidelines in Appendix A.

Requirement	
ST 1204.1-26	When extracting a sub-window of Motion Imagery Data from an existing Motion Imagery Data set, the resulting Motion Imagery Data shall contain a copy of the original Foundational Identifier, if one exists, with a Window Identifier included.

6.1.5 Foundational Core Identifier Implementation

Two special cases reduce the processing during identification insertion: **Pass-back** and **Pre-fill**.

In **Pass-back** the Platform Identifier (Physical or Virtual) passes “backwards” to the Sensor in real time, so the Sensor can construct a complete Foundational Core Identifier. This can eliminate the need for processing within the platform, because the Platform Identifier insertion occurs within the Sensor. For example, if the Sensor produces compressed Motion Imagery then, by using Pass-back, the platform does not need to update the Foundational Core Identifier; the Platform Identifier information inserts into the Motion Imagery during the compression process within the Sensor.

Pre-fill refers to a situation where a Sensor produces a Foundational Core Identifier with a temporary ID for the Platform Identifier. Replacing this temporary ID will not affect the bandwidth of the Motion Imagery (e.g., this prevents the need for de-multiplexing and then re-multiplexing a MPEG-2 Transport Stream). The value for the temporary ID is the nil UUID, which is 16 bytes of the hex value “0x00” or “0x00000000000000000000000000000000”.

Requirement(s)	
ST 1204.1-31	When a MIIS conformant Sensor pre-fills the Foundational Core Identifier with a temporary value for the Platform Identifier, 0x00000000000000000000000000000000 shall be the only value used.
ST 1204.1-32	When Foundational Core Identifiers within Motion Imagery data leave the platform or are stored on the platform, they shall be fully formed with no temporary Identifiers.
ST 1204.2-45	The Foundational Core Identifier shall be a UUID generated following the guidelines in Appendix A.

6.2 Minor Core Identifier

The purpose of the **Minor Core Identifier** is to support a low level of identification when Foundational Core Identifiers are absent; generating and inserting a Minor Core Identifier into a Motion Imagery stream or file occurs downstream post the collection process. A Minor Core Identifier is a UUID primarily for legacy systems and devices, where a legacy device or system is one that does not support the Foundational Core Identifier.

There are no requirements for when a Minor Core Identifier changes, so the system implementation determines when to switch to a new Minor Core Identifier. For example, if a distribution system detects a completed mission or a new mission started, this might warrant a new Minor Core Identifier for that mission. Alternatively, a new Minor Core Identifier could cover a specific time interval (i.e., every twelve hours). There is no guarantee a Minor Core Identifier will change to match missions or source changes and, because they are inserted downstream from some users of the Motion Imagery, they are considered inadequate to satisfy the four problems listed in Section 1 (for all users).

Requirement(s)	
ST 1204.1-27	When Motion Imagery Data does not contain a Foundational Core Identifier and the Motion Imagery Data has been transmitted from the platform, a MIIS conformant system shall create a Minor Core Identifier and insert it into the Motion Imagery Data.
ST 1204.1-28	When Motion Imagery Data includes a Foundational Core Identifier then a Minor Core Identifier shall NOT be used, generated or inserted into the Motion Imagery Data.
ST 1204.1-29	Sensors and/or Platforms shall generate only Foundational Core Identifiers.
ST 1204.1-30	The Minor Core Identifier shall be a UUID generated following the guidelines in Appendix A.

6.3 MIIS Core Identifier Summary

Figure 6 illustrates the complete structure of the MIIS Core Identifier, which includes Version and Usage Values (“Vers+Usage”) for the Core Identifier.

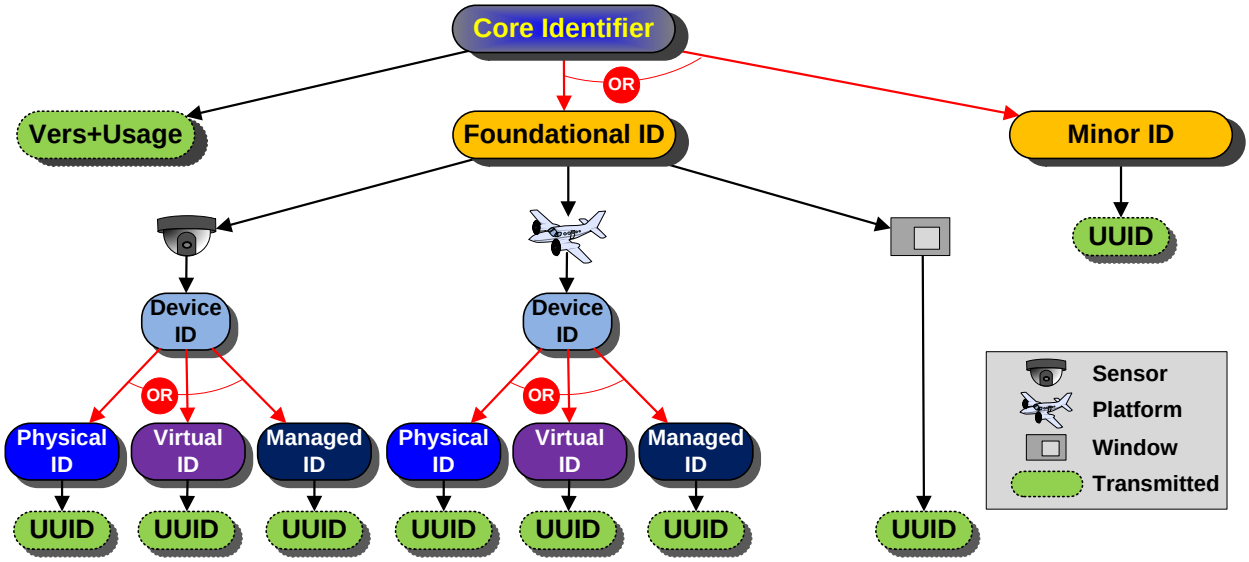

Figure 6: Complete MIIS Core Identifier

Although Figure 6 shows the full hierarchy of MIIS Core Identifier structure, the only data potentially transmitted are the values in green.

A Physical Foundational Core Identifier is a Foundational Core Identifier with both a Physical Sensor Identifier and Physical Platform Identifier. A Physical Foundational Core Identifier will provide the most benefit to end users, so the primary goal of this standard is to have all systems produce them. Because of implementation costs and other issues, a Virtual Identifier or Managed Identifier-based Foundational Core Identifier is acceptable until a Physical Foundational Core Identifier is implementable. Minor Core Identifiers only apply when the source system cannot generate a Foundational Core Identifier.

7 Core Identifier Implementation

In addition to the raw Identifier Component UUID values, the Core Identifier includes additional information indicating the Version of the Core Identifier and Identifier quality information of each Identifier Component.

This section provides the required and recommended implementation details for the Motion Imagery Identification System (MIIS) - Core Identifier.

Section 7.1 describes the MIIS Core Identifier by using Extended Backus–Naur Form (EBNF) (defined in ISO/IEC 14977 [2]), which is independent of the transmission format.

Section 7.4 supplies the details for how to represent the EBNF structure in different formats including KLV (Key-Length-Value) [3], text, and XML (Extensible Markup Language) [4].

7.1 Core Identifier - EBNF

The following shows the structural EBNF for the MIIS Core Identifier:

FCID = Foundational Core Identifier
MCID = Minor Core Identifier

```
Core Identifier     = Version, Usage Value, FCID | MCID;
Version             = Version number of the Core Identifier structure (currently = 1)
Usage Value         = ((Sensor-ID-Type, Platform-ID-Type, Window-ID-Type, 'None') | ('None',
                      'None', 'None', Minor-ID-Type)) - ('None', 'None', 'None', 'None');
Sensor-ID-Type      = 'Physical' | 'Virtual' | 'Managed' | 'None';
Platform-ID-Type    = 'Physical' | 'Virtual' | 'Managed' | 'None';
Window-ID-Type      = 'Included', 'None';
Minor-ID-Type       = 'Included', 'None';
FCID                = (Sensor ID, [Platform ID], [Window ID]) | (Platform ID, [Window ID]) |
                      Window ID;
Sensor ID           = UUID;
Platform ID         = UUID;
Window ID           = UUID;
UUID                = 16 byte Universally Unique Identifiers (UUIDs) created following the
                      guidelines in Appendix A
MCID                = UUID;
```

7.2 Version

The Version of the Core Identifier is a number that aligns with the EBNF structure for the Core Identifier. The version number is always the first value in the Core Identifier so parsers can read this value and determine how to interpret the remaining Core Identifier values. Currently, the Core Identifier Version is 1. Prior versions of this standard assigned the version number based on the document version of this standard. This is no longer the practice. Thus, changes to this standard which are non-material to the Core Identifier structure will remain compatible with prior versions of the standard.

7.3 Identifier Quality Information and Usage Value

A **Usage Value** represents the combined identifier quality information for the Sensor Identifier and Platform Identifier along with Window Identifier and Minor Identifier information (in that order). For the Sensor Identifier and Platform Identifier there are four possible Identifier Component qualities to assign: Physical, Virtual, Managed or None. For the Window Identifier and Minor Identifier there are only two possible values: Included or None. Table 2 lists the usage values for each Identifier Component.

Table 2: Usage Value Components

Sensor Identifier Quality Value	Platform Identifier Quality Value	Window Identifier	Minor Identifier
Physical	Physical	Included	Included
Virtual	Virtual	None	None
Managed	Managed		
None	None		

7.3.1 Usage Value Mapping

Section 7.1 shows the Core Identifier EBNF structure and its various elements. The Usage Value is common to both types of transmission – binary and text discussed in the following sections.

The Core Identifier Usage Value maps the desired selection of choices in the Usage Value EBNF structure into a one-byte (1) BER-OID encoded value indicated in Table 3. Future changes, if needed, will use bit 7 to indicate an expanded length of this value from one byte to multiple bytes. All future changes will retain the other values in the Usage Value to maintain backward compatibility.

Table 3: Usage Value

Bit(s)	Meaning	Bit Values
7 - MSB	Reserved	Always Zero
6,5	Sensor-ID-Type	11 = Physical, 10=Virtual, 01=Managed, 00=None
4,3	Platform-ID-Type	11 = Physical, 10=Virtual, 01=Managed, 00=None
2	Window-ID-Type	1=Included, 0=None
1	Minor-ID-Type	1=Included, 0=None
0 - LSB	Reserved	Always Zero

Example 1: (bits b7 and b0 always set to 0 shown in red)

	Sensor-ID-Type	Platform-ID-Type	Window-ID-Type	Minor-ID-Type
Identifier Quality	Physical	Physical	None	None
Mapped bits b7-b0	011xxxx0	0xx11x0	0xxxx0x0	0xxxxxx00
Usage Value	Combining bits b7-b0 = 0 11 11 0 0 0 = 0111 1000 = 0x78			

Example 2: (bits b7 and b0 always set to 0 shown in red)

	Sensor-ID-Type	Platform-ID-Type	Window-ID-Type	Minor-ID-Type
Identifier Quality	Physical	Virtual	Included	None
Mapped bits b7-b0	011xxxx0	0xx10x0	0xxxx1x0	0xxxxxx00
Usage Value	Combining bits b7-b0 = 0 11 10 1 0 0 = 0111 0100 = 0x74			

7.4 Core Identifier Formats

The Core Identifier has two different styles of formats: binary and textual.

Table 4 identifies the different formats and their reference document sections.

Table 4: Core Identifier Formats

Format Name	Transmission Style	Section	Description
KLV	Binary	7.4.1.1	Used as KLV metadata within a Motion Imagery stream or file
Textual Format	Text	7.4.2.1	Used for operator readability
XML	Text	7.4.2.2	Used when communicating identity information outside of a Motion Imagery stream or file (i.e., system-to-system)

7.4.1 Binary Format

The Binary Format compacts the Core Identifier to as few bits as possible for transmission through limited bandwidth data channels. The principle use for the binary format is coding for KLV. Table 5 shows how the mapping for the Core Identifier EBNF to a Core Identifier binary value.

Table 5: Core Identifier EBNF to Core Identifier Binary Value Mapping

EBNF	KLV Construct
Core Identifier	Block of data, which is the combination of Version, Usage Value and FCID or MCID. Values as shown in the rest of this table
Version	Version is a BER-OID (see [5]) encoded value for the version number
Usage Value	Bitwise mapping of the Usage Value as shown in Table 3
FCID	Combination of Sensor ID, Platform ID and/or Window ID; variable length depending on the included values. Each UUID value is 16 bytes so valid lengths are 16, 32 or 48 bytes. The order of the Sensor ID, Platform ID and Window ID is important and should follow the EBNF in Section 7.1
Sensor ID	Single UUID value of 16 bytes
Platform ID	Single UUID value of 16 bytes
Window ID	Single UUID value of 16 bytes
MCID	Single UUID value of 16 bytes

Figure 7 illustrates an example of a Foundational Core Identifier Binary Value. The development of Usage Byte 0x70 in the figure is similar to the development of the Usage Bytes shown in Examples 1 and 2 above.

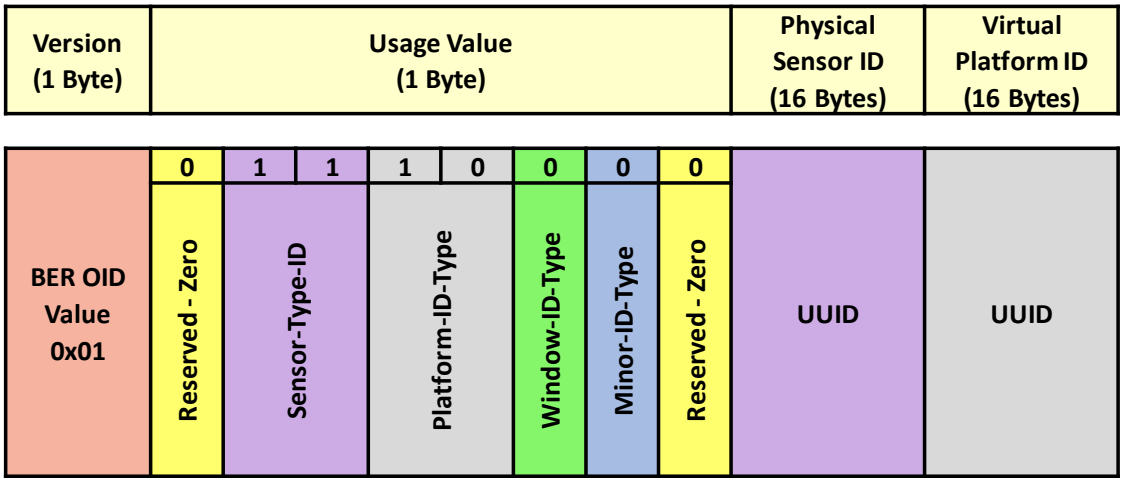

Figure 7: Foundational Core Identifier Example

The top row identifies the byte assignments which includes a Version byte, a Usage Value byte, a 16-byte Physical Sensor Identifier (ID), and a 16-byte Virtual Platform Identifier (ID). The second row is the information represented by the bytes/bits. The one-byte Version is set to 0x01 corresponding to version one (1) of the Core Identifier EBNF. The one-byte Usage Value bits are set as follows (left to right): Reserved bit 7 is set to zero; both bit 6 and 5 are one to indicate a Physical Sensor Identifier (according to Table 3) ; bit 4 and 3 are set to one and zero, respectively, to indicate a Virtual Platform Identifier (see Table 3); bit 2 is set to zero to indicate that a Window Identifier is NOT included; bit 1 is set to zero to indicate that a Minor Identifier is NOT included; Reserved bit 0 is set to zero. The Physical Sensor Identifier and Virtual Platform Identifier are blocks of binary UUID values.

The following is an example of the binary-encoded Foundational Core Identifier for Figure 7.

Raw Hex Values of Binary Format Example

```
0170 F592 F023 7336 4AF8 AA91 62C0 0F2E B2DA 16B7 4341 0008 41A0 BE36
5B5A B96A 3645
```

Here the Version byte (0x01) combines with the Usage Byte (0x70) to form the leading two-byte value of 0170 hex.

Table 6 breaks out the raw hex values according to their defined positions in Figure 7.

Table 6: Explanation of Binary Format Hex Example

	Version	Usage Value Byte								UUID - 1	UUID - 2
Hex	01	70								F592 F023 7336 4AF8 AA91 62C0 0F2E B2DA	16B7 4341 0008 41A0 BE36 5B5A B96A 3645
Meaning	Version 1	Bit	0	1	1	1	0	0	0	Physical Sensor ID (UUID)	Virtual Platform ID (UUID)
		Bit Meaning	Reserved (Always 0)	11 Indicates that the first UUID is a Sensor ID and it's a Physical ID		10 Indicates that the second UUID is a Platform ID and it's a Virtual ID		0 Indicates no Window ID included	0 Indicates no Minor ID included	Reserved (Always 0)	

The string format for this Foundational Core Identifier example is show in Section 7.4.2.1.

7.4.1.1 Core Identifier KLV Format

The Core Identifier as a binary value maps directly into KLV constructs including as a standalone value, in sets and packs. As a standalone value the binary value Core Identifier prefixes a 16-byte Core ID Key and BER-encoded Length, as shown in Figure 8.

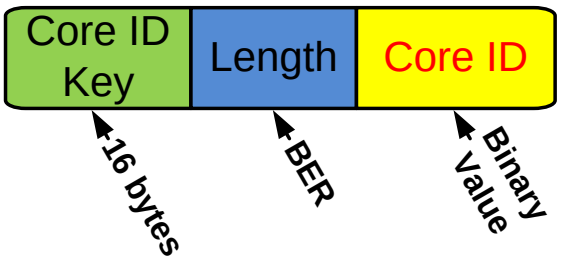

Figure 8: Standalone Core Identifier in KLV

The Core ID Key for a standalone value is 0x060E-2B34-0101-0101-0E01-0405-0300-0000 (CRC 30280) and the normative Symbol name is "core_id" (registered in MISB ST 0807 [6]).

The pack is a floating length pack and the Core Identifier binary value is the last element in the pack because the length of the Core Identifier binary value cannot be known a priori. The following is an example of a Foundational Core Identifier encoded as KLV with its explanation expressed in Table 7. Section 7.4.2.1 illustrates the string format for this Foundational Core Identifier example. Appendix E provides additional examples for the Binary Format for KLV.

Raw Hex Values of KLV Example

```
060E2B3401010101 0E01040503000000 220170F592F02373 364AF8AA9162C00F
2EB2DA16B7434100 0841A0BE365B5AB9 6A3645
```

Table 7: Explanation of KLV Hex Example

	KLV Key	KLV Length	Binary Value (Hex)
Hex	060E2B3401010101 0E01040503000000	22	0170 F592 F023 7336 4AF8 AA91 62C0 0F2E B2DA 16B7 4341 0008 41A0 BE36 5B5A B96A 3645
Meaning	KLV Key for Core Identifier	Value is 34 bytes	Core Identifier Binary Value with Version, Usage Byte and UUID values - See Table 6

The Core Identifier needs to be present in the stream periodically.

Requirement	
ST 1204.1-34	The MIIS conformant System shall insert a Core Identifier into the KLV stream at least every 30 seconds or when the value changes (i.e., sensor changes).

MISB standards supply guidance on how to carry KLV metadata in the various classes of Motion Imagery as described in the MISP [7]. MISP approved containers include Serial Digital Interface (SDI), MPEG-2 Transport Stream, GigE Vision, etc. for transmitting KLV metadata. Refer to the MISP for approved containers and supporting standards' documents which provide guidance on how to insert KLV metadata into a desired container.

7.4.2 Textual Format

The Textual Format provides a human readable Core Identifier value for end users. The Core Identifier Textual Format (defined in the next section) is the format used by the Core Identifier XML Format.

7.4.2.1 Core Identifier Text Format

When communicating Core Identifiers via text, the text format aids in human readability and data re-entry. Using this format promotes consistency and reduces data entry error when displaying or printing a Core Identifier for a user.

In accordance with [8], to aid in readability and reduce entry mistakes the textual format separates the Core Identifier value into groups of hex digits with each group a maximum of four hex characters long. Each group has a separator character – either a colon, ':', dash, '-', or slash,'/', depending on the context of the group. The four-character grouping structure includes grouping the values for the embedded UUID components of the Core Identifier. This UUID format is slightly different than the canonical UUID format; however, to convert an embedded UUID into the UUID Hexadecimal Representation ([9] section 6.4) the first and last two separator characters are removed. Additionally, an added check value enables a system to validate if a user has entered or copied a value correctly. Table 8 describes the mapping between the Core Identifier EBNF to Core Identifier Textual Format:

Table 8: Core Identifier EBNF to Core Identifier Text Format Mapping

EBNF	Textual Construct
Core Identifier	Single line of characters which is the combination of Usage Value and FCID or MCID values as shown in the rest of this table
Version	Version is a Hex value of the version number
Usage Value	Usage Value is a Hex value of the information shown in Table 3. A colon character ':' follows the Usage Value
FCID	Combination of Sensor ID, Platform ID and/or Window ID with each value separated by a slash character, '/'. The order of the Sensor ID, Platform ID and Window ID is important and should follow the EBNF in Section 7.1. A colon character ':' follows the last ID
Sensor ID	UUID String Value
Platform ID	UUID String Value
Window ID	UUID String Value
MCID	UUID String Value
UUID String Value	39-character value composed of: hex value of the UUID, plus separator characters (after every four hex characters insert a dash '-') e.g., 0102-0304-0506-0708-090A-0B0C-0D0E-0F10
Another Note	After the complete string of hex characters, append a check hex value. Note the check character validates only the hex digits, not the separator characters. See Appendix B for the check value algorithm.

The following are abbreviated notations for FCID and MCID, respectively:

VersionUsageValue:SensorID/PlatformID/WindowID:CheckValue

VersionUsageValue:MCID:CheckValue

The following is an example of a text formatted Foundational Core Identifier with Table 9 explaining the values.

Example Core Identifier Text Value

```
0170:F592-F023-7336-4AF8-AA91-62C0-0F2E-B2DA/16B7-4341-0008-41A0-BE36-5B5A-B96A-3645:D3
```

Appendix E provides additional examples for the Text Format.

Table 9: Explanation of Core Identifier Text Example

	Version	Usage Value Byte	Colon	UUID-1	Slash	UUID-2	Colon	Check Value
Hex	01	70	:	F592-F023-7336-4AF8-AA91-62C0-0F2E-B2DA	/	16B7-4341-0008-41A0-BE36-5B5A-B96A-3645	:	D3
Meaning	Version 1	Same as Usage Value Byte in Table 6	Separator	Physical Sensor ID	Separator	Virtual Platform ID	Separator	Check Value for Hex characters

Note: If a UUID value is extracted from the string format it can be converted to the UUID Hexadecimal Representation ([9] *section 6.4) by removing the first and last two separator characters (i.e., dashes). For example:*

ST 1204 Format	0102-0304-0506-0708-090A-0B0C-0D0E-0F10
UUID Hexadecimal Representation	01020304-0506-0708-090A-0B0C0D0E0F10

7.4.2.2 Core Identifier in XML

The XML schema in Listing 1 is the format for the Core Identifier expressed as XML. The Core Identifier Text Format (see Section 7.4.2.1) uses the MiisCoreId XML tag and the CoreIdType provides validation of the format of the Core Identifier.

Listing 1 - Core Identifier XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.nga.gov/MiisSchema"
xmlns:tns="http://www.nga.gov/MiisSchema"
elementFormDefault="qualified">
<xs:element name="MiisCoreId" type="tns:CoreIdType" />
<xs:simpleType name="CoreIdType"  >
  <xs:restriction base="xs:string">
    <xs:pattern
      value="[A-F0-9]{4}:(([A-F0-9]{4}-){7}[A-F0-9]{4})(/([A-F0-
9]{4}-){7}[A-F0-9]{4})?(/([A-F0-9]{4}-){7}[A-F0-9]{4})?:[A-F0-
9]{2}">
    </xs:pattern>
  </xs:restriction>
</xs:simpleType>
</xs:schema>
```

Listing 2 shows an example of the XML generated from the Listing 1 schema for the Core Identifier.

Listing 2 - XML Example of Core Identifier

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<MiisCoreId xmlns="http://www.nga.gov/MiisSchema">
0170:F592-F023-7336-4AF8-AA91-62C0-0F2E-B2DA/16B7-4341-0008-41A0-
BE36-5B5A-B96A-3645:D3
</MiisCoreId>
```

Requirement	
ST 1204.1-39	When transmitting a Core Identifier within XML, the Core Identifier XML Format shall be used.

8 Unique Identifier for Frames

To generate a unique identifier for a Motion Imagery frame, combine the Core Identifier with a MISP Time System timestamp: either a Precision Time Stamp or a Nano Precision Time Stamp. Together these two pieces of information provides a universally unique identifier for every frame across all Motion Imagery Data sources. Table 10 illustrates an example where a Core Identifier plus a MISP Timestamp forms a unique identifier for Frame 1 and Frame 2.

Table 10: Unique Identifier for Motion Imagery Frames

Frame	Core Identifier	MISP Timestamp
Frame 1	0170:F592-F023-7336-4AF8-AA91-62C0-0F2E-B2DA/16B7-4341-0008-41A0-BE36-5B5A-B96A-3645:D3	2012-08-20T13:23:13.134000
Frame 2	0170:F592-F023-7336-4AF8-AA91-62C0-0F2E-B2DA/16B7-4341-0008-41A0-BE36-5B5A-B96A-3645:D3	2012-08-20T13:23:13.167000

9 Generating Enterprise UUID's from Core Identifiers

Core Identifiers find many different purposes, one being an identifier within enterprise systems. The Core Identifier can be either an FCID or MCID each of which has their own technique for generating enterprise identifiers. For an MCID the enterprise identifier is simply the UUID within the MCID. The MCID-based enterprise identifier does provide some functionality; however, it is limited in scope to where the MCID was inserted into the stream - see Appendix C for a discussion on this issue.

For FCIDs up to three enterprise identifiers are possible: (1) an identifier from the FCID's Sensor Identifier, (2) an identifier from the FCID's Platform Identifier and (3) an identifier from the FCID's Window Identifier. They are usable independently or together. Since the Core Identifier can change over time, combining the three identifiers into one UUID is not a viable option as an Enterprise UUID.

See Appendix C and Appendix D for more information on the configurations of various systems and how they relate to the resulting identifiers.

10 Identification Retention

Motion Imagery Data is subject to modification as it travels from the sensor to users through various stages of processing. It is important to maintain the identification of the data throughout all processing. The result of Motion Imagery exploitation is to generate products. In order to ensure traceability back to the original source, it is critical for identification information to stay with the product.

Requirement(s)	
ST 1204.1-40	When a system or software manipulates, transcodes, extracts or augments the Motion Imagery, it shall preserve the Core Identifier in both the altered Motion Imagery and metadata.
ST 1204.1-41	All Motion Imagery products generated from the Motion Imagery shall retain the Core Identifier from the source Motion Imagery.

11 Compliance Levels

Enumerating the possible combinations of physical/virtual sensors and platforms along with the Motion Imagery extracted from a window either with or without a Window Identifier, provides a developer or acquisition program an ability to specify which combination meets their application needs. Each combination is assigned a Compliance Level for the MIIS Core Identifier as shown Table 11 and Table 12.

Table 11 lists Compliance Levels for all systems. Systems which deliver Motion Imagery from an extracted window are further qualified according to Table 12.

Table 12 lists the Compliance Levels for systems extracting a window and further conditioned on whether the system supplies a Window Identifier. A Compliance Level is lower for a system that extracts a window from Motion Imagery Data but does NOT add a Window Identifier into the Core Identifier (Level 17 down through Level 2).

In either table the higher the Compliance Level the closer the overall system adheres to this standard. The goal is that all systems achieve a Compliance Level of 32.

Table 11: MIIS Core identifier Compliance Levels

Level	Sensor	Platform
32	Physical	Physical
31	Physical	Virtual
30	Physical	Managed
29	Virtual	Physical
28	Virtual	Virtual
27	Virtual	Managed
26	Managed	Physical
25	Managed	Virtual
24	Managed	Managed
23	Physical	None
22	Virtual	None
21	Managed	None
20	None	Physical
19	None	Virtual
18	None	Managed
17-2	Window Based Levels – See Table 12	
1	Minor Identifier	
0	No Identification	

Table 12: Compliance Levels for Windowing Systems

Level	Sensor	Platform	Window ID	Comment
32	Physical	Physical	Yes	Motion Imagery from Window
31	Physical	Virtual	Yes	Motion Imagery from Window
30	Physical	Managed	Yes	Motion Imagery from Window
29	Virtual	Physical	Yes	Motion Imagery from Window
28	Virtual	Virtual	Yes	Motion Imagery from Window
27	Virtual	Managed	Yes	Motion Imagery from Window
26	Managed	Physical	Yes	Motion Imagery from Window
25	Managed	Virtual	Yes	Motion Imagery from Window
24	Managed	Managed	Yes	Motion Imagery from Window
23	Physical	None	Yes	Motion Imagery from Window
22	Virtual	None	Yes	Motion Imagery from Window
21	Managed	None	Yes	Motion Imagery from Window
20	None	Physical	Yes	Motion Imagery from Window
19	None	Virtual	Yes	Motion Imagery from Window
18	None	Managed	Yes	Motion Imagery from Window
17	None	None	Yes	Motion Imagery from Window
16	Physical	Physical	No	Motion Imagery is from a Window but no Window ID
15	Physical	Virtual	No	Motion Imagery is from a Window but no Window ID
14	Physical	Managed	No	Motion Imagery is from a Window but no Window ID
13	Virtual	Physical	No	Motion Imagery is from a Window but no Window ID
12	Virtual	Virtual	No	Motion Imagery is from a Window but no Window ID
11	Virtual	Managed	No	Motion Imagery is from a Window but no Window ID
10	Managed	Physical	No	Motion Imagery is from a Window but no Window ID
9	Managed	Virtual	No	Motion Imagery is from a Window but no Window ID
8	Managed	Managed	No	Motion Imagery is from a Window but no Window ID
7	Physical	None	No	Motion Imagery is from a Window but no Window ID
6	Virtual	None	No	Motion Imagery is from a Window but no Window ID
5	Managed	None	No	Motion Imagery is from a Window but no Window ID
4	None	Physical	No	Motion Imagery is from a Window but no Window ID
3	None	Virtual	No	Motion Imagery is from a Window but no Window ID
2	None	Managed	No	Motion Imagery is from a Window but no Window ID

12 Deprecated Requirements

The following are requirements specific to the carriage of metadata within SMPTE Serial Digital Interface (SDI) containers. Because the Core Identifier is metadata and the MISP and other

MISB standards prescribe requirements for the carriage of metadata, these deprecated requirements are redundant. Furthermore, as MISB approves other container technologies, rather than add additional requirements in this document to track these new containers, the encompassing requirements in the MISP and its support standards provide enough guidance.

Requirement(s)	
ST 1204.1-33 (Deprecated)	Where additional formats are used to transmit Motion Imagery, they shall have the MIIS Core Identifier imbedded within both the uncompressed imagery frame data and the metadata (including LVMI formats).
ST 1204.1-35 (Deprecated)	When a MIIS Compliant Sensor uses SMPTE ST 292 the Core Identifier shall be inserted into the VANC data space of every Motion Imagery frame according to MISB ST 0605.
ST 1204.1-36 (Deprecated)	When a MIIS Compliant Sensor uses SMPTE ST 424 the Core Identifier shall be inserted into the VANC data space of every Motion Imagery frame according to MISB ST 0605.
ST 1204.1-37 (Deprecated)	When Core Identifiers from the VANC data space are moved to a MPEG-2 Transport Stream's metadata stream and the Core Identifier changes the Core Identifier shall be immediately inserted into the resulting metadata stream.
ST 1204.1-38 (Deprecated)	When Core Identifiers from the VANC data space are moved to a MPEG-2 Transport Stream's metadata stream and the Core Identifier does not change the Core Identifier shall be repeated at least every 30 seconds in the resulting metadata stream.

Appendix A UUID Generation - Normative

A Universal Unique IDentifier (UUID) can be generated using multiple techniques as discussed in [9] and [10]. NGA has written a Recommended Practice NGA.RP.0001 [11] that discusses allowed UUID versions.

UUID Version 1 uses the Ethernet MAC address and system clock in implementation; version 4 relies on a random number generator; version 5 uses SHA hash coding of some textual information.

This standard recommends, when possible, to construct UUID's from unique device information through the version 5 SHA hash coding process, for example combining manufacturer name, model number, serial number and use the SHA hash function to generate the UUID.

If using version 4, ensure to provide a good cryptographic random number generator; refer to [12] for information on random number generation.

MISB does not recommend Version 1 UUIDs, but, if necessary, generate them only if the device has a MAC address.

Sometimes it is necessary to form a new UUID from two or more existing UUID's. The following technique enables the possibility of reconstructing a UUID from the source UUIDs. To combine the UUIDs convert each of the UUIDs into the hexadecimal format defined in Section 6.4 of [9] in uppercase (e.g., F81D4FAE-7DEC-11D0-A765-00A0C91E6BF6). Then append together (without separator characters) and perform the version 5 hash code algorithm.

Requirement(s)	
ST 1204.1-42	UUIDs shall be generated using only UUID versions 1, 4 and 5 as stated in NGA.RP.0001.
ST 1204.1-43	The creator of the UUID shall ensure the information (or combination of information) used as the hash source is unique for every device.
ST 1204.1-44	The null MAC address shall not be used for UUID generation.

Appendix B Check Value

A check value added to the data validates a transfer of characters. Applying the same algorithm used in computing the check value, a receiver of the data can compare it to the original check value. If the values are the same, the data most likely transmitted correctly (these checks are not perfect); otherwise there was an error in the transmission. The method used, in this standard for computing the check value is based on [13] [14], which is for hexadecimal values only. The method specifically detects common problems that occur when entering data manually, which is different than other techniques such as checksums and CRC's designed for burst errors related to transmission issues.

The algorithm in [13] computes a four-bit check digit based on a set of permutations of the hex digits. This standard makes two adjustments to this algorithm. First, in [13] computing the check value relies on inverting the "sum" of the permutations. This step is unnecessary, so the check value for this standard is just the "sum". The second adjustment is a second set of permutations

and running the algorithm a second time to compute a second four-bit check digit. Then, combine both check digits into a single byte as the check value. The extra run provides a better check value and uses a full byte for the check value. To improve efficiency both check digits compute in a single loop through the hex values.

Notation:

- ^ is xor
- [a,b,c,d] denotes the four bits of an integer value ranging from 0 to 15. For example, 5 = [0,1,0,1].
- p (lower case) is a permutation of a hex value, noted p(H) = p ([a,b,c,d])
- $p^j(H)$ is multiple permutations of hex value H. Example: $p^2(H) = p(p(H))$
- P(0:3) = {0,1, 2,3} means array items zero through three are set to 0, 1, 2, 3, respectively

The algorithm has two parts, first a table initialization then a check value computation.:

B.1 Table Initialize

Initializing the permutation table is a one-time operation by either computing the permutations or statically defining the array in **Table 13**. Using the bit manipulation method from [13], create two 2-dimensional permutation tables P and Q:

P(0, 0:15) = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15}

For all j=0 to 15 and all i = 1 to 14 P(i,j) = pMap(P(i-1,j))

For all j=0 to 15 and all i = 1 to 14 Q(i,j) = qMap(P(i-1,j))

pMap(i)=[a^b,c,d,a] (Example: pMap(13) = pMap([1,1,0,1]) = [1^1,0,1,1]=[0,0,1,1]=3 (0x3))

qMap(i)=[d,a^d,b,c] (Example: qMap(13) = qMap([1,1,0,1]) = [1,1^1,1,0]=[1,0,1,0]=10 (0xA))

Note: qMap is not in [13].

The P and Q permutation should equal the values in **Table 13**.

Table 13: P and Q Permutations

	P Permutations																Q Permutations															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	0	2	4	6	8	A	C	E	9	B	D	F	1	3	5	7	0	C	1	D	2	E	3	F	4	8	5	9	6	A	7	B
2	0	4	8	C	9	D	1	5	B	F	3	7	2	6	A	E	0	6	C	A	1	7	D	B	2	4	E	8	3	5	F	9
3	0	8	9	1	B	3	2	A	F	7	6	E	4	C	D	5	0	3	6	5	C	F	A	9	1	2	7	4	D	E	B	8
4	0	9	B	2	F	6	4	D	7	E	C	5	8	1	3	A	0	D	3	E	6	B	5	8	C	1	F	2	A	7	9	4
5	0	B	F	4	7	C	8	3	E	5	1	A	9	2	6	D	0	A	D	7	3	9	E	4	6	C	B	1	5	F	8	2
6	0	F	7	8	E	1	9	6	5	A	2	D	B	4	C	3	0	5	A	F	D	8	7	2	3	6	9	C	E	B	4	1
7	0	7	E	9	5	2	B	C	A	D	4	3	F	8	1	6	0	E	5	B	A	4	F	1	D	3	8	6	7	9	2	C
8	0	E	5	B	A	4	F	1	D	3	8	6	7	9	2	C	0	7	E	9	5	2	B	C	A	D	4	3	F	8	1	6
9	0	5	A	F	D	8	7	2	3	6	9	C	E	B	4	1	0	F	7	8	E	1	9	6	5	A	2	D	B	4	C	3
10	0	A	D	7	3	9	E	4	6	C	B	1	5	F	8	2	0	B	F	4	7	C	8	3	E	5	1	A	9	2	6	D
11	0	D	3	E	6	B	5	8	C	1	F	2	A	7	9	4	0	9	B	2	F	6	4	D	7	E	C	5	8	1	3	A
12	0	3	6	5	C	F	A	9	1	2	7	4	D	E	B	8	0	8	9	1	B	3	2	A	F	7	6	E	4	C	D	5
13	0	6	C	A	1	7	D	B	2	4	E	8	3	5	F	9	0	4	8	C	9	D	1	5	B	F	3	7	2	6	A	E
14	0	C	1	D	2	E	3	F	4	8	5	9	6	A	7	B	0	2	4	6	8	A	C	E	9	B	D	F	1	3	5	7

B.2 Check Value Computation

To compute the check value, loop through all the individual hex characters and process through the permutation tables as follows:

Let hexString be the hex string to compute the check value for (e.g., 031FA3)

set pCheck=0

set qCheck=0

loop i through hexString (starting i=1)

pCheck=pCheck^P(i % 15, hexString(i))

qCheck=qCheck^P(i % 15, hexString(i))

end loop

checkVal = Shift Left 4 bits (pCheck) or'ed with (qCheck)

For example, using 031FA3 the result is 0x79.

Iterations:

i	Hex Char	P(i%15, hexChar)	pCheck	Q(i%15,hexChar)	qCheck
--	N/A	N/A	0x0	N/A	0x0
1	0	P(1,0) = 0x0	0x0	Q(1,0) = 0x0	0x0
2	3	P(2,3) = 0xC	0xC	Q(2,3) = 0xA	0xA
3	1	P(3,1) = 0x8	0x4	Q(3,1) = 0x3	0x9
4	F	P(4,F) = 0xA	0xE	Q(4,F) = 0x4	0xD

5	A	P(5,A) = 0x1	0xF	Q(5,A) = 0xB	0x6
6	3	P(6,3) = 0x8	0x7	Q(6,3) = 0xF	0x9

Combining the last two highlighted values by shifting the pCheck to the left by four bits, results in a check value of 0x79.

Appendix C Core Identifier Background

Figure 9 shows a high-level view of data flow from a single Motion Imagery Data source (1) distributed to a group of users (4). In this example there are two primary routes for distributing the Motion Imagery to end users, either a **Direct Route** or via a **Control Station Route** (2). With the **Direct Route** the Motion Imagery travels directly to an end user (A) without passing through the Ground Control Station (GCS) or any other node(s). While there is only one Direct Route end user in this example, there could be a large set of Direct Route end users. With the **Control Station Route** the Motion Imagery passes through the control station and then to a distribution network (3) where it routes to the end users (B), (C), (D) and (E).

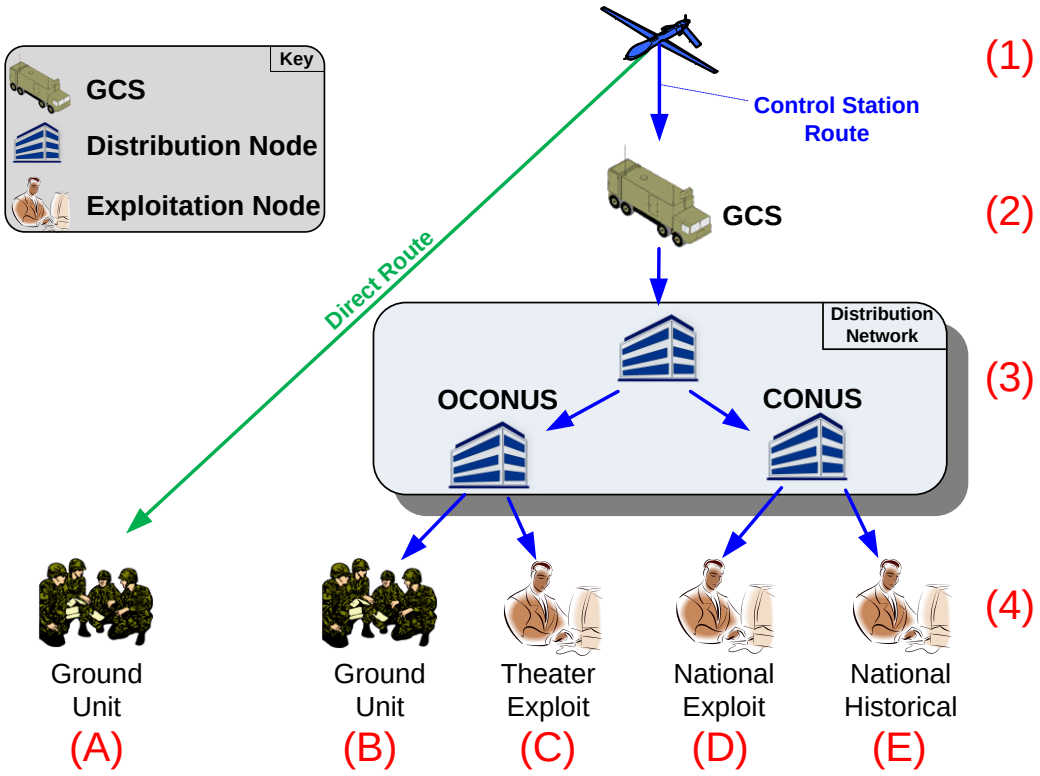

Figure 9: Motion Imagery Data Flow Scenario

Four points along the Motion Imagery Data flow offer opportunity to insert identification information: (1) the source, (2) the GCS, (3) the distribution cloud, or (4) the end user sites. The optimal point to insert identification information is at the source (1), because the identifier(s) will

be available (i.e., flow down) to all the components of the overall system (i.e., distribution network and end-user sites). If insertion of identifiers cannot occur at the source, then the next best place is within the GCS (2). If insertion of identifiers cannot occur in the GCS then the next possible insert point is within the distribution network (3); however, the usefulness of this varies depending on which "part" of the network the insertion occurs. If inserting the identifiers immediately after receiving the Motion Imagery Data (i.e., right after the GCS (2)) then this provides a consistent identifier for all downstream dissemination. If inserting the identifiers just before leaving the distribution network, then there is no guarantee that all copies of the Motion Imagery Data leaving the network will have the same identifiers, which, does not provide the consistency desired. If neither the source (1) nor GCS (2) nor distribution network (3) inserts an identifier, then the user sites (4) can insert their own identifiers; however, each site would have different identifiers unless there is some extra backchannel coordination performed between sites.

Identification **coverage** is the percentage of users who can access or use a common identifier. The goal is to provide an identifier with 100% coverage, so it is available to all users of the Motion Imagery Data. Table 14 approximates the coverage based on where the insertion of identifiers occurs in the Motion Imagery Data.

Table 14: Coverage Example for Figure 9

Insertion Point	Sites using the same ID					Coverage	Comments
	A	B	C	D	E		
Source (1)	Y	Y	Y	Y	Y	5/5 = 100%	All users using the same Identifier; this provides perfect coverage and is the goal of this standard
GCS (2)	N	Y	Y	Y	Y	4/5 = 80%	Most users would be receiving the identification information
CONUS Node (3)	N	N	N	Y	Y	2/5 = 40%	Reduces user coordination between CONUS and OCONUS users; alternate methods are necessary
User Site (4)	N	N	N	N	Y	1/5 = 20%	Only users at site (4) would have a common Identifier so the identification information is only affective in the one location

Figure 9 also highlights a problem with the value to insert as the identifier. As discussed previously, if the identifier is not available at the source then the identifier cannot provide *100% or full* coverage. When a "downstream" component (e.g., GCS) inserts an identifier, it cannot provide *full* coverage which means some users of the system will have an identifier, and some will not. For the parts of the system which do not have identifiers, other "downstream" components could insert different identifiers.

For example, in Figure 9, if the OCONUS component inserts an identifier independently from the CONUS component, each would be inserting a different identifier in which case theater exploiters (or systems) and national exploiters (or systems) would not be able to recognize they may be operating on the same stream. This promotes the need for a method to qualify the

goodness or quality of the identifier. Inserting identifiers at the source (1) results in a high-quality identifier. Likewise, inserting identifiers at the GCS (2) results in a lower-quality identifier. Finally, inserting identifiers at network distribution or user sites results in a minimal-quality identifier. Another factor affecting identifier quality is the identification information inserted into the Motion Imagery Data.

At the Motion Imagery source there are several options for providing a unique identifier. Appendix D illustrates a couple of source architectures. It is clear there is no one single piece of information usable as an identifier, so a collection of identifiers acts as the source identifier. Ideally, every device through which Motion Imagery passes should add identifying information to the stream creating an "ID history"; however, this is impractical, since it would require that every device would have to de-multiplex, update and re-multiplex the Motion Imagery stream.

This standard defines a minimum set of elements such that the identifiers still provide uniqueness and a basis for augmentation data. The minimum identifier is a Sensor Identifier combined with a Platform Identifier; these two pieces of information uniquely identify a source. However, when a sensor system, such as a LVMI sensor, produces one or more "virtual" windows from a Motion Imagery source, an additional identifier called a Window Identifier supplements these identifiers. These two or three elements of identifying information form the Foundational Core Identifier. Since the generation of a Foundational Core Identifier uses known information from the platform and sensor the only insertion points can be either the Source (1) or the Control Station (2). Thus, the quality of the Foundational Core Identifier is high. All other insertion points cannot positively ensure a high-quality sensor/platform identifier. At such points, when inserting an identifier, use a random Universal Unique Identifier (UUID). This type of identifier called a Minor Core Identifier provides minimal quality.

Note that a Foundational Core Identifier can change over time within a mission. For example, when the sensor source changes (i.e., EO to IR) the sensor component of the identifier will change; however, the platform component of the identifier remains the same. It is up to the receivers of the data to interpret these values and determine how to process and use them. The Foundational Core Identifier along with the MISP timestamp enables frame-by-frame unique identifiers which can be a basis for Augmentation Identifiers.

In summary, the only option that guarantees 100% identification coverage is inserting identifiers at the Source or Control Station. Unfortunately, currently deployed systems do not have this capability, so downstream or minimal identification methods will need to suffice until such time that the platforms/sensors provide source-based identification information (i.e., Foundational Core Identifiers).

Appendix D Architecture Overview of Platform Sensor

This appendix describes a general architecture for Motion Imagery systems along with several examples of the general architecture.

Figure 10 shows a generic architecture to support a single sensor and the data flow through the architecture. This figure shows four different types of items: data producers (blue), data processors/movers (yellow), data storage (orange) and users (green). Table 15 provides a description of each item.

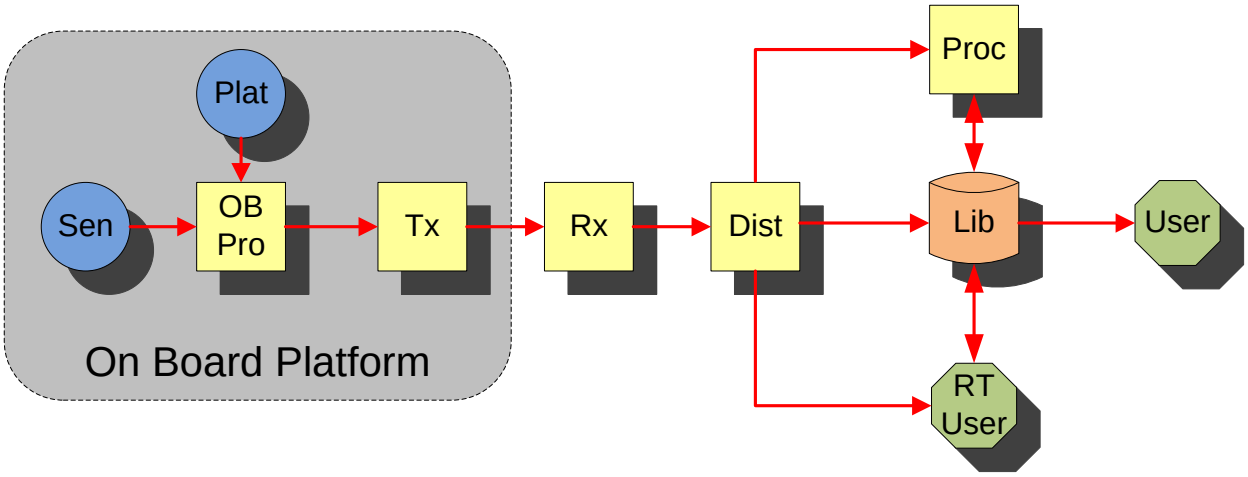

Figure 10: Generic Motion Imagery Architecture

Table 15: Architecture Components

Item	Name	Description
Sen	Sensor	Collects Motion Imagery (including metadata) from scene
Plat	Platform	Physically supports the sensor, produces metadata (e.g., orientation, etc.)
OBPro	On Board Processing	Processes Motion Imagery and metadata onboard the platform. Processing examples: Encoder, Multiplexor, Switching, on-board storage, etc.
Tx	Transmitter	Transmits Motion Imagery to one or more Receivers – can include encoding and/or transrating of Motion Imagery
Rx	Receiver	Receives transmitted Motion Imagery – can include multiplexing additional metadata into Motion Imagery
Dist	Distribution	Dissemination of Motion Imagery through one or more networks. Can include switches, guards, SATCOMs, etc.
Lib	Library	Library storage of Motion Imagery Data
Proc	Processing	Real-time and Post-processing of Motion Imagery Data
RTUser	Real Time User	User exploiting Motion Imagery Data during mission. Operations include annotation, clipping, etc.
User	User	Non-Real time exploiting Motion Imagery mission

Motion Imagery systems contain at least one sensor, so other items in this list may not be present. For example, a hand-held camera has a sensor and on-board storage where the user can view the Motion Imagery directly from the camera system. In this case the Tx, Rx, Dist, Lib, and Proc are unnecessary. Another example is an airborne system using all items in the list/diagram. This generic architecture applies to all Motion Imagery systems used in Air, Ground, Surface and Space.

To understand the complexities of the identification problem a couple of detailed examples follow below. Figure 11 illustrates a UAS architecture/data flow with two sensors and an A/B switch to select which sensors data transmits to the receivers. Two different encoders compress

the selected stream, one for SATCOM transmission and one for line–of-site (LOS) transmission. Each encoder may be operating at different data rates and qualities.

Given this architecture, inserting an identifier before the encoders is the only way to ensure that the SATCOM users and LOS users both receive the same identifier. In this example a unique Platform Identifier solves the identification problem since the system supports one mission; however, other examples show that this is not always the case.

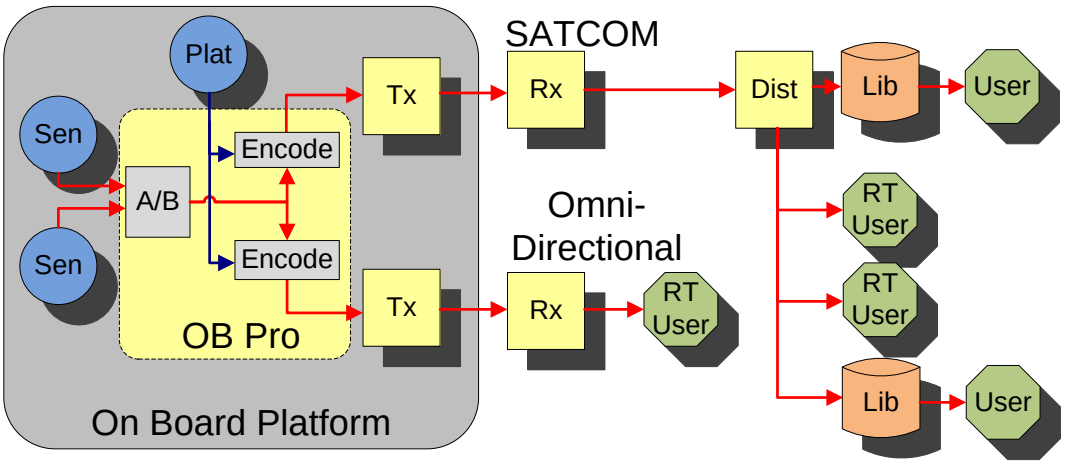

Figure 11: UAS Dual Sensor with Dual Encoder

Figure 12 illustrates an example of many sensors whose data path switches (as needed) to a specific transmitter. Each sensor might support different missions or user needs.

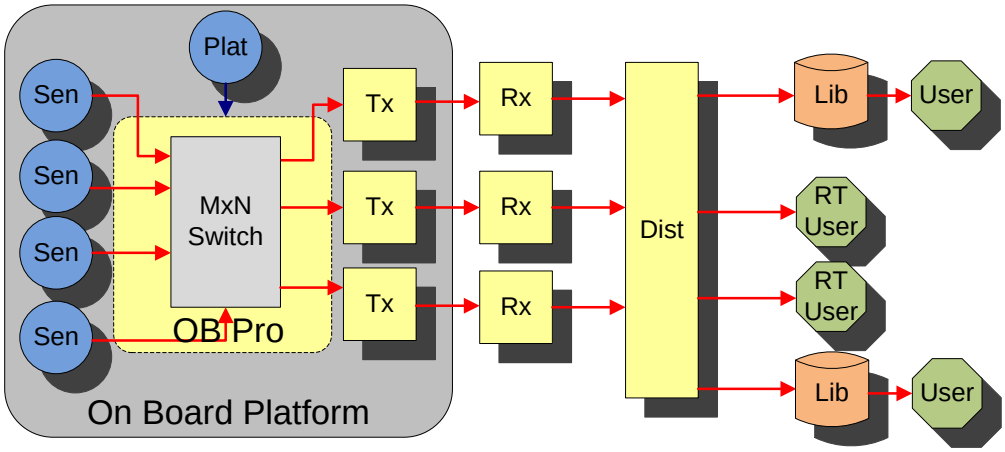

Figure 12: Multi-Sensor Multi-Transmission

In this example a Platform Identifier is insufficient, and the more appropriate identifier is a Sensor Identifier.

Appendix E Example Core Identifiers

The following are examples of various Core Identifier configurations. Each example shows a Binary Format for KLV and a Textual Format mapping.

Example E1: Foundational Core ID with Sensor ID, Platform ID and Window ID

```
0154:C7D1-6253-98A2-41C2-BA6E-90F8-FCC7-3914/E047-AB3E-81BE-41ED-9664-09B0-2F44-
5FAB/5E71-B0DC-20FE-4920-8216-26D6-4F61-D863:C8

Usage Byte = 54 = 0 10 10 1 0 0 = (2=VIRTUAL,2=VIRTUAL,1=Included,0=None)

Version (1)

KLV: 060E2B3401010101 0E01040503000000 320154C7D1625398 A241C2BA6E90F8FC
C73914E047AB3E81 BE41ED966409B02F 445FAB5E71B0DC20 FE4920821626D64F 61D863

Text: 0154:C7D1-6253-98A2-41C2-BA6E-90F8-FCC7-3914/E047-AB3E-81BE-41ED-9664-09B0-2F44-
5FAB/5E71-B0DC-20FE-4920-8216-26D6-4F61-D863:C8
```

Example E2: Foundational ID with Sensor ID and Platform ID

```
0150:08CE-252E-D0FA-49E3-B1A0-65E6-1D57-20DC/EAF1-8A27-A086-4019-A586-EAAF-9715-
7BFA:DA

Usage Byte = 50 = 0 10 10 0 0 0 = (2=VIRTUAL,2=VIRTUAL,0=None,0=None)

Version (1)

KLV: 060E2B3401010101 0E01040503000000 22015008CE252ED0 FA49E3B1A065E61D
5720DCEAF18A27A0 864019A586EAAF97 157BFA

Text: 0150:08CE-252E-D0FA-49E3-B1A0-65E6-1D57-20DC/EAF1-8A27-A086-4019-A586-EAAF-9715-
7BFA:DA
```

Example E3: Foundational ID with physical Sensor ID and managed Platform ID

```
0168:F354-666E-D552-4C0D-9168-A745-4CEB-A073/840A-4799-BBC0-4BD5-97A7-56F6-4092-
AF7B:AA

Usage Byte = 68 = 0 11 01 0 0 0 = (3=PHYSICAL,1=MANAGED,0=None,0=None)

Version (1)

KLV: 060E2B3401010101 0E01040503000000 220168F354666ED5 524C0D9168A7454C
EBA073840A4799BB C04BD597A756F640 92AF7B

Text: 0168:F354-666E-D552-4C0D-9168-A745-4CEB-A073/840A-4799-BBC0-4BD5-97A7-56F6-4092-
AF7B:AA
```

Example E4: Foundational ID with physical Sensor ID and physical Platform ID

```
0178:865E-FD9C-EF8A-41C3-8244-B885-AFCC-40BF/ED8A-9AB8-72E2-4165-9979-7E5A-F54A-
5B9A:25

Usage Byte = 78 = 0 11 11 0 0 0 = (3=PHYSICAL,3=PHYSICAL,0=None,0=None)

Version (1)

KLV: 060E2B3401010101 0E01040503000000 220178865EFD9CEF 8A41C38244B885AF
CC40BFED8A9AB872 E2416599797E5AF5 4A5B9A

Text: 0178:865E-FD9C-EF8A-41C3-8244-B885-AFCC-40BF/ED8A-9AB8-72E2-4165-9979-7E5A-F54A-
5B9A:25
```

Example E5: Foundational ID with Sensor ID and Window ID

```
0144:340F-463B-AEC2-4F8C-BD45-92AE-DE80-E1C6/F0AF-8673-3A17-424C-B060-3EE4-A86B-
38F9:D9

Usage Byte = 44 = 0 10 00 1 0 0 = (2=VIRTUAL,0=None,1=Included,0=None)

Version (1)

KLV: 060E2B3401010101 0E01040503000000 220144340F463BAE C24F8CBD4592AEDE
80E1C6F0AF86733A 17424CB0603EE4A8 6B38F9

Text: 0144:340F-463B-AEC2-4F8C-BD45-92AE-DE80-E1C6/F0AF-8673-3A17-424C-B060-3EE4-A86B-
38F9:D9
```

Example E6: Foundational ID with Sensor ID

```
0140:BC76-CFEF-0BEE-41A4-9618-EB4B-010D-2F08:B6

Usage Byte = 40 = 0 10 00 0 0 0 = (2=VIRTUAL,0=None,0=None,0=None)

Version (1)

KLV: 060E2B3401010101 0E01040503000000 120140BC76CFEF0B EE41A49618EB4B01 0D2F08

Text: 0140:BC76-CFEF-0BEE-41A4-9618-EB4B-010D-2F08:B6
```

Example E7: Foundational ID with Platform ID and Window ID

```
0114:3D50-2DB6-4A93-44C3-B56E-94AD-7C4E-E476/45E2-8FAF-C2D3-4A6E-815B-5FE6-B0A9-
6ABD:F1

Usage Byte = 14 = 0 00 10 1 0 0 = (0=None,2=VIRTUAL,1=Included,0=None)

Version (1)

KLV: 060E2B3401010101 0E01040503000000 2201143D502DB64A 9344C3B56E94AD7C
4EE47645E28FAFC2 D34A6E815B5FE6B0 A96ABD

Text: 0114:3D50-2DB6-4A93-44C3-B56E-94AD-7C4E-E476/45E2-8FAF-C2D3-4A6E-815B-5FE6-B0A9-
6ABD:F1
```

Example E8: Foundational ID with Platform ID

```
0110:1AB8-231E-17E8-4748-A133-CE93-89A7-A060:25

Usage Byte = 10 = 0 00 10 0 0 0 = (0=None,2=VIRTUAL,0=None,0=None)

Version (1)

KLV: 060E2B3401010101 0E01040503000000 1201101AB8231E17 E84748A133CE9389 A7A060

Text: 0110:1AB8-231E-17E8-4748-A133-CE93-89A7-A060:25
```

Example E9: Foundational ID with Window ID

```
0104:C2A7-D724-96F7-47DB-A23D-A297-3007-5876:55

Option Byte = 04 = 0 00 00 1 0 0 = (0=None,0=None,1=Included,0=None)

Version (1)

KLV: 060E2B3401010101 0E01040503000000 120104C2A7D72496 F747DBA23DA29730 075876

Text: 0104:C2A7-D724-96F7-47DB-A23D-A297-3007-5876:55
```

Example E10: Minor ID

```
0102:03DD-9DEE-FB48-477B-8204-B050-6F6B-2A33:25
Usage Byte = 02 = 0 00 00 0 1 0 = (0=None,0=None,0=None,1=Included)
Version (1)
KLV: 060E2B3401010101 0E01040503000000 12010203DD9DEEFB 48477B8204B0506F 6B2A33
Text: 0102:03DD-9DEE-FB48-477B-8204-B050-6F6B-2A33:25
```